EMBEDDED OPEN SOURCE SUMMIT

EMBEDDED LINUX CONFERENCE

Clock and Power Management in ARM SCMI

2024-04-16, Seattle

Dhruva Gole

Kamesh Gurudasani

Texas Instruments

Disclaimers

- This is a technology presentation, not product-readiness or roadmap commitment
- Opinions presented here are that of the speakers and may not reflect that of Texas Instruments Inc.

About us: TI Processors and Open source

Decades of contribution and collaboration

Ingrained culture to give back to the community

Upstream FIRST!

Focus on long term, sustainable and quality products

Upstream and opensource ecosystem in device architecture

Upstream FIRST mentality!

Introduction to the Speakers

Dhruva Gole

(Linux Kernel Engineer at Texas Instruments)

- Power Management on Sitara SOC’s
- U-Boot, TF-A, Linux device driver development

Kamlesh Gurudasani

(*Linux Kernel Engineer at Texas Instruments)*

- Security and Power management on Sitara SOC’s
- U-boot, TF-A, OPTEE, Linux device drivers development

Overview

Background

Basics of Power and Clock Management

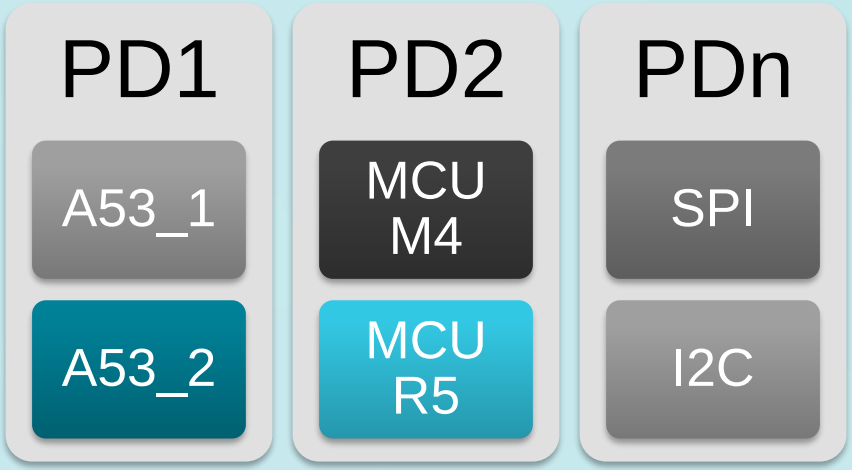
TF-A, Linux and SCMI

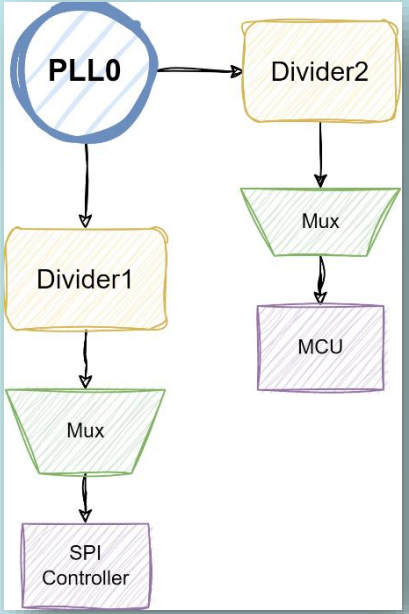
Implementing ARM SCMI

Background

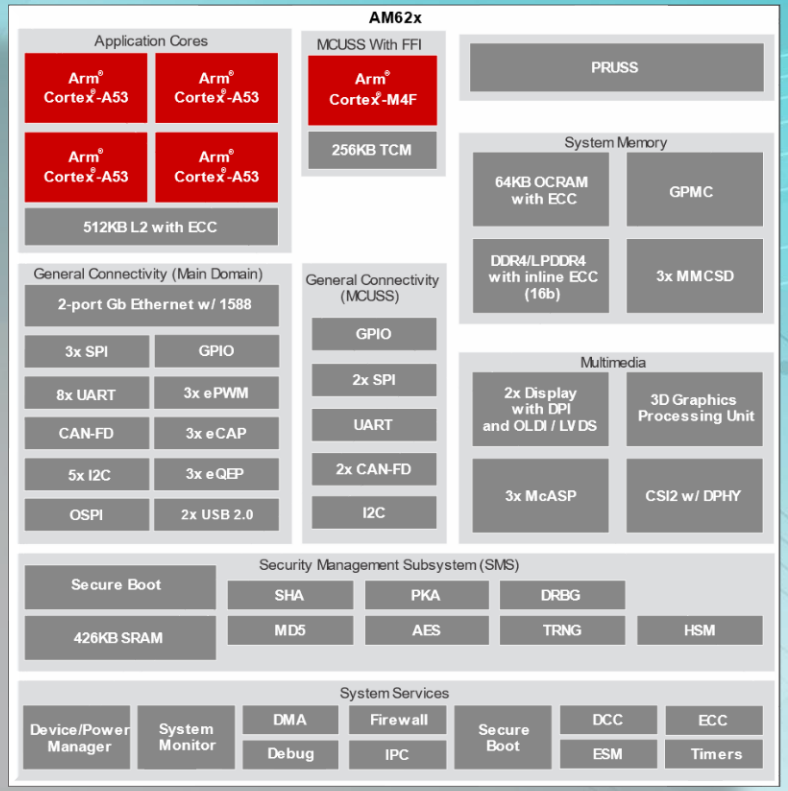
Basics of Power and Clock Management

- Each SoC is subdivided into power domains
- These domains can be turned on /off based on requirement.
- Any entity in the SoC can require a component from any power domain.
- Hence they can simply the central Firmware entity to power on/off the domain.



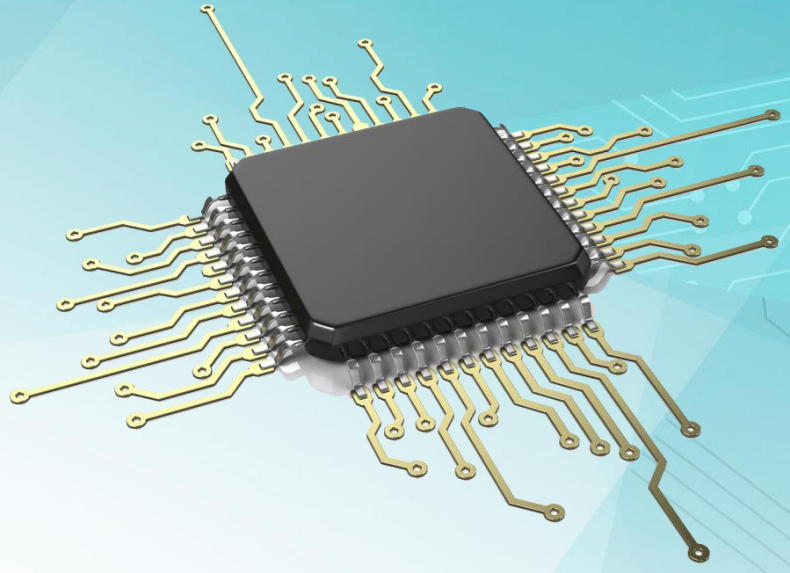


Clock Management



https://www.ti.com/product/AM625

- Multiple ARM Based MCU and MPU cores in a single SoC
- Linux running on A53
- MCU Firmware (Free RTOS/ Zephyr) running on cortex M4F MCU core.
- Device Management (DM) Firmware running on cortex R5 core. This is central entity responsible for clock/ power mgmt.
- How will they all talk to DM for requesting clock / power?



Need for standard protocol

- One of the major players – TI came up with it's own "TI SCI" protocol for solving the challenge above.
- Any entity in SoC could request for clock and power from the *Device Manager (DM)*.
- Could be Linux/ MCU FW following this protocol to request for resources

firmware: Add support for TI System Control Interface (TI-SCI) protocol driver

From:	Nishanth Menon <nm-AT-ti.com>
To:	Russell King <rmk+kernel-AT-armlinux.org.uk>, Sudeep Holla <sudeep.holla-AT-arm.com>, Santosh Shilimkar <ssantosh-AT-kernel.org>
Subject:	[PATCH 0/5] firmware: Add support for TI System Control Interface (TI-SCI) protocol driver
Date:	Fri, 19 Aug 2016 17:51:54 -0500
Message-	<20160819225159.10758-1-nm@ti.com>

https://lwn.net/Articles/697909/

- ARM soon after introduced the **ARM System Control and Management Interface (SCMI)**
- Similar to DM FW: ARM recommends SCP-firmware (a software reference implementation)
- This central entity can even be Trusted Firmware-ARM in the case of Linux centric devices.

firmware: ARM System Control and Management Interface(SCMI) support

From:	Sudeep Holla <sudeep.holla-AT-arm.com>
To:	ALKML <linux-arm-kernel-AT-lists.infradead.org>, LKML <linux-kernel-AT-vger.kernel.org>, DTML <devicetree-AT-vger.kernel.org>
Subject:	[PATCH 0/9] firmware: ARM System Control and Management Interface(SCMI) support
Date:	Mon, 26 Jun 2017 16:55:04 +0100
Message-ID:	<1498492513-29771-1-git-send-email-sudeep.holla@arm.com>

- https://lore.kernel.org/all/1519403030-21189-1-git-send-email-sudeep.holla@arm.com/
- https://lwn.net/Articles/726475/

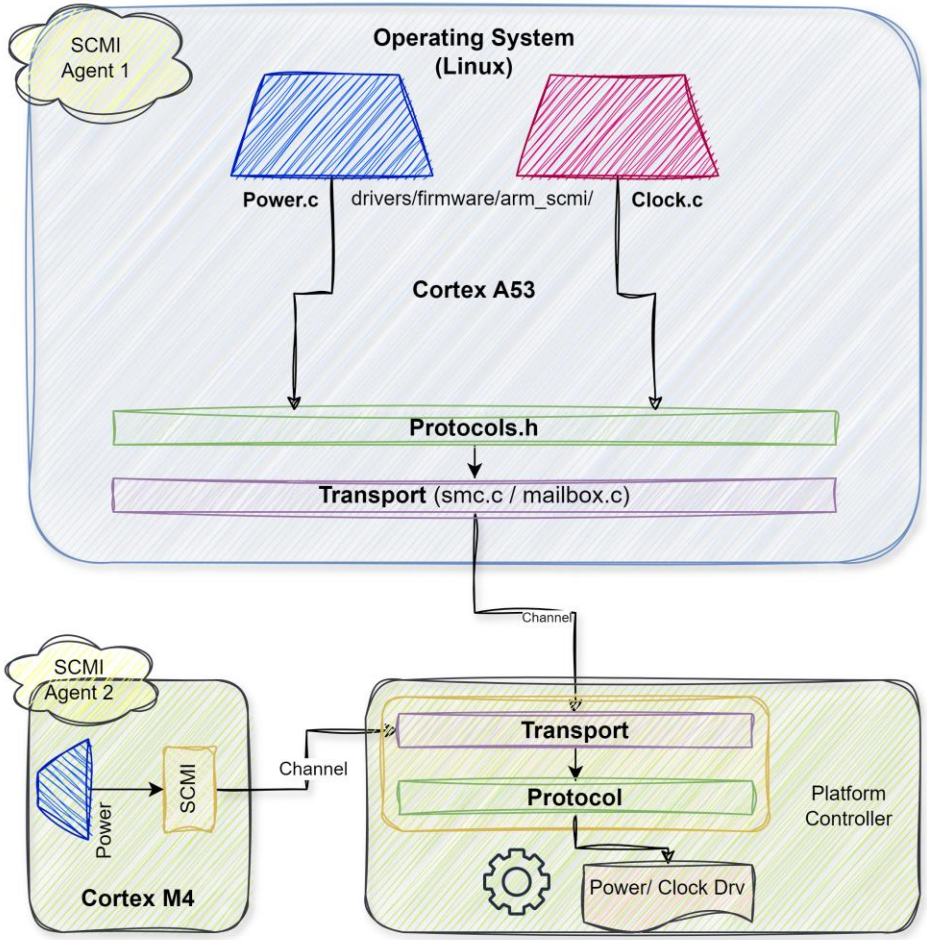
Performance domain management

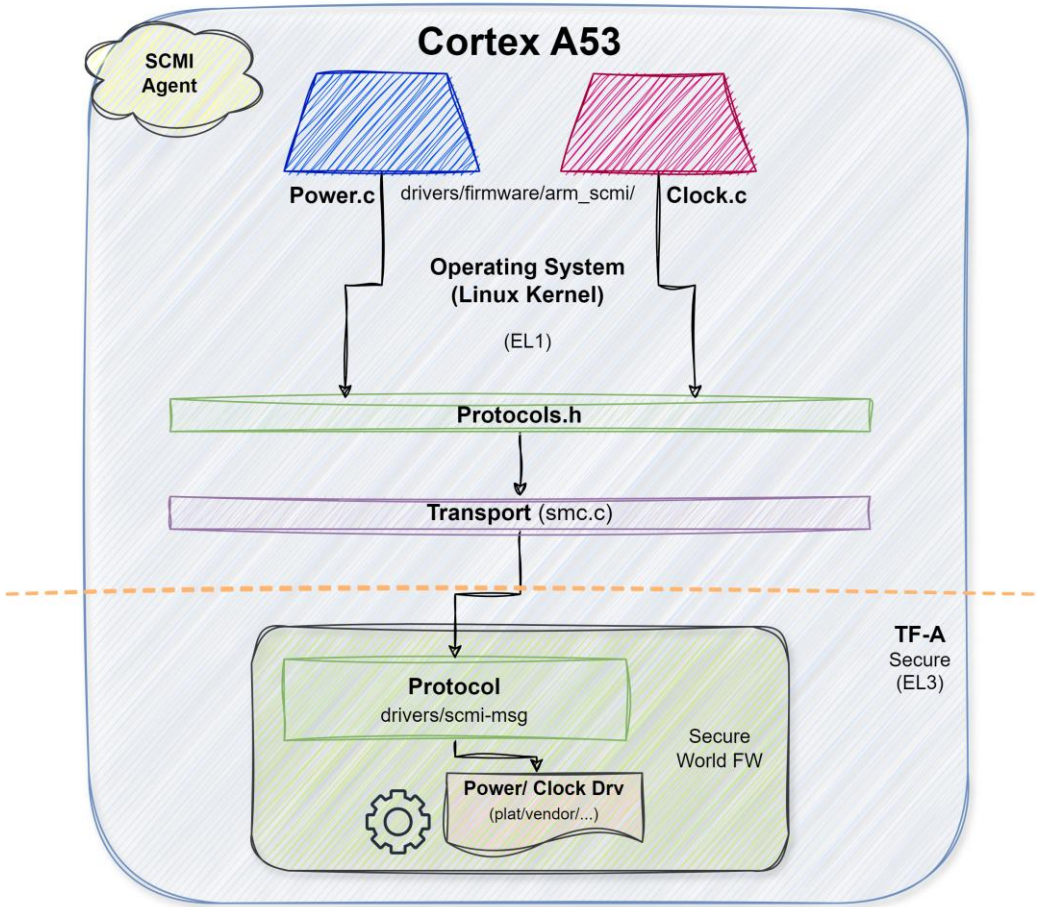
- Ability to control the performance of a domain that is composed of compute engines
- For eg. application processors (APs), GPUs, or other accelerators.
- Set of devices that always have to run at the same performance level
- Operates on an abstract integer performance scale
- Values in the scale might represent actual frequencies

Additional Protocols Supported

- Power domain
- System PM
- Performance
- Clocks
- Reset Domain
- Voltage
- Pin control

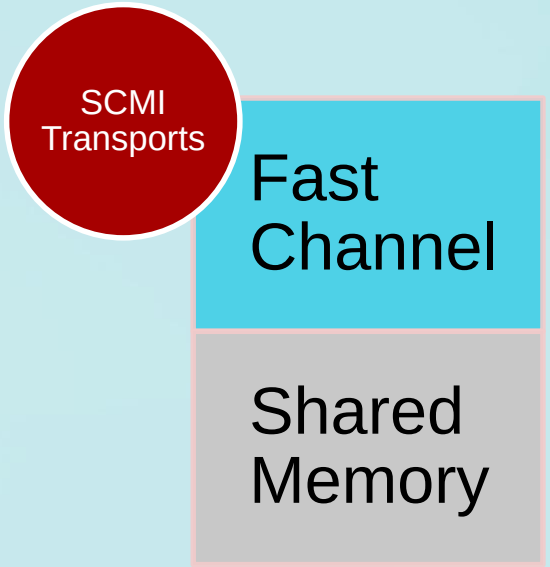
TF-A, Linux and SCMI


SCMI
Agent 1
Operating System
(Linux)
Power.c
drivers/firmware/arm_scmi/
Clock.c
Cortex A53
Protocols.h
Transport (smc.c / mailbox.c)
Channel
SCMI
Agent 2
Power
SCMI
Channel
Cortex M4
Transport
Protocol
Platform
Controller
Power/ Clock Drv


Cortex A53
SCMI Agent
Power.c
drivers/firmware/arm_scmi/
Clock.c
Operating System (Linux Kernel)
(EL1)
Protocols.h
Transport (smc.c)
TF-A
Secure
(EL3)
Protocol
drivers/scmi-msg
Secure World FW
Power/ Clock Drv
(plat/vendor/...)

How can Linux talk to TF-A?

- **Linux** is non secure (EL1 privilege)
- TF-A runs at EL3: the highest privilege level
- SMC Calling Convention Interface (ARM DEN 0028A) is gateway



Using SCMI on your devices

Terminology Used

- **Agent** is used to describe the caller of the System Control & Management Interface.
- **Platform** describes the set of hardware components that interpret the messages and provide the necessary functionality
- **Transport** describes the method by which protocol messages are communicated between agents and the platform

SCMI support in TF-A

- Not every protocol has to be present in an implementation
- Platform chooses which protocols it exposes to a given agent.
- The Base protocol is the only one which **must** be implemented
  - Used by agent to discover which protocols are available.

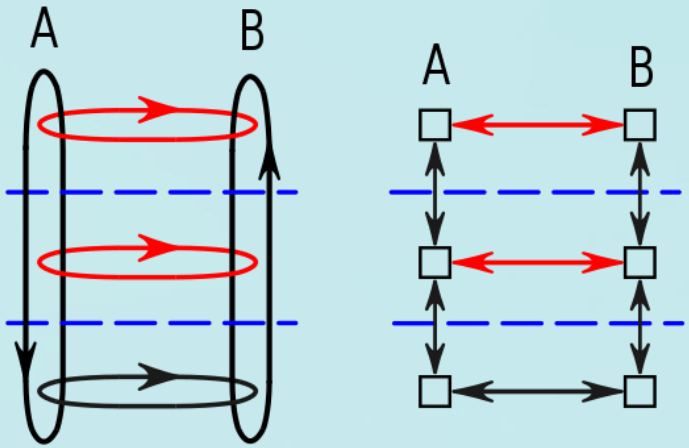


Table 2 Protocol identifiers *v3.2

protocol_id	Description
0x0 - 0xF	Reserved
0x10	Base protocol
0x11	Power domain management protocol
0x12	System power management protocol
0x13	Performance domain management protocol
0x14	Clock management protocol
0x15	Sensor management protocol
0x16	Reset domain management protocol
0x17	Voltage domain management protocol
0x18	Power capping and monitoring protocol
0x19	Pin Control protocol
0x1A-0x7F	Reserved for future use by this specification
0x80-0xFF	Reserved for vendor or platform-specific extensions to this interface

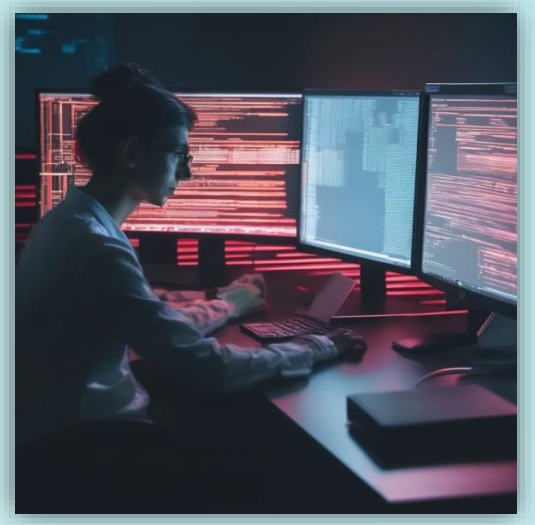
- TF-A will require scmi drivers
- What are the platform specific drivers that you'll need to add?

SCMI

- [ ] Clock
- [ ] Power
- [ ] Performance

SVC

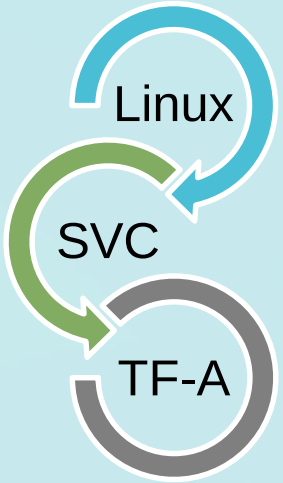
- [ ] sip_smc_handler
- [ ] runtime service descriptor



Implementation

SVC (Supervisor Call)

- The SVC instruction is used to perform system calls and other critical low-level functions needed by applications.
- Registers are 'shared' information between worlds



```
/* Define a runtime service descriptor for fast SMC calls */

DECLARE_RT_SVC(k3_svc,
               OEN_SIP_START,
               OEN_SIP_END,
               SMC_TYPE_FAST,
               NULL,
               sip_smc_handler);
```

```
/*
 * This function is responsible for handling all SiP calls from the NS world
 */
uintptr_t sip_smc_handler(uint32_t smc_fid,
                          u_register_t x1,
                          u_register_t x2,
                          u_register_t x3,
                          u_register_t x4,
                          void *cookie,
                          void *handle,
                          u_register_t flags)
```

SCMI server

```
static struct scmi_msg_channel scmi_channel[] = {
  [0] = {
    .shm_addr = 0x70000000,
    .shm_size = 0x100,
  },
};
```

Called from
drivers/scmi-msg/base.c

```c
// platform init function
scmi_smt_init_agent_channel(&scmi_channel[0]);

const char *plat_scmi_vendor_name(void)
{
  return vendor;
}

const char *plat_scmi_sub_vendor_name(void)
{
  return sub_vendor;
}

size_t plat_scmi_protocol_count(void)
{
    [...]
    return count;
}

const uint8_t *plat_scmi_protocol_list(unsigned int agent_id)
{
  // Can be more!
  return SCMI_PROTOCOL_ID_CLOCK;
}


struct scmi_msg_channel *plat_scmi_get_channel(unsigned int agent_id)
{
  assert(agent_id < ARRAY_SIZE(scmi_channel));
  return &scmi_channel[agent_id];
}
```

Power Domains

- Protocol must at least support the ON and OFF states but can support additional power states
- Protocol commands take integer identifiers to identify the power domain.
- Clock/ SCMI ID of devices should start from 0..N and can’t have gaps in between

plat/vendor/
…/scmi_pd.c

```
size_t plat_scmi_pd_count(unsigned int agent_id __unused)

const char *plat_scmi_pd_get_name(unsigned int agent_id __unused,
          unsigned int pd_id __unused)

unsigned int plat_scmi_pd_statistics(unsigned int agent_id __unused,
             unsigned long *pd_id __unused)

unsigned int plat_scmi_pd_get_attributes(unsigned int agent_id __unused,
           unsigned int pd_id __unused)

unsigned int plat_scmi_pd_get_state(unsigned int agent_id __unused,
            unsigned int pd_id __unused)

int32_t plat_scmi_pd_set_state(unsigned int agent_id __unused,
             unsigned int flags __unused,
             unsigned int pd_id __unused,
             unsigned int state __unused)
```

Clocking in TF-A

- Enable or disable a clock.
- Set the rate and other configuration of the clock synchronously or asynchronously
- Get and set the parent of a clock.
- Allows to change the parent clock if needed

plat/vendor/…/scmi_clock.c

```
size_t plat_scmi_clock_count(unsigned int agent_id __unused)

const char *plat_scmi_clock_get_name(unsigned int agent_id __unused,
            unsigned int scmi_id __unused)

int32_t plat_scmi_clock_rates_array(unsigned int agent_id __unused,
           unsigned int scmi_id __unused,
           unsigned long *rates __unused,
           size_t *nb_elts __unused,
           uint32_t start_idx __unused)

unsigned long plat_scmi_clock_get_rate(unsigned int agent_id __unused,
              unsigned int scmi_id __unused)

int32_t plat_scmi_clock_set_rate(unsigned int agent_id __unused,
         unsigned int scmi_id __unused,
         unsigned long rate __unused)

int32_t plat_scmi_clock_get_state(unsigned int agent_id __unused,
          unsigned int scmi_id __unused)

int32_t plat_scmi_clock_set_state(unsigned int agent_id __unused,
          unsigned int scmi_id __unused,
          bool enable_not_disable __unused)
...
```

Linux Support

```
&cbass_main {
  oc_sram: sram@70000000 {
    compatible = "mmio-sram";
    reg = <0x00 0x70000000 0x00 0x10000>;
    #address-cells = <1>;
    #size-cells = <1>;
    ranges = <0x0 0x00 0x70000000 0x10000>;

    scmi_shmem: sram@0 {
      compatible = "arm,scmi-shmem";
      reg = <0x0 0x100>;
    };
  };
```

- Shared memory region is required for communication with TF-A
- This shared memory region can be setup as shown.

`smc-id` dictates how an SMC call goes through

```
scmi: scmi {
  compatible = "arm,scmi-smc";
  arm,smc-id = <0x82000010>;
  shmem = <&scmi_shmem>;
  #address-cells = <1>;
  #size-cells = <0>;

  scmi_clk: protocol@14 {
    reg = <0x14>;
    #clock-cells = <1>;
  };

  scmi_pd: protocol@11 {
    reg = <0x11>;
    #power-domain-cells = <1>;
  };
};
```

How do I decide `smc-id`?

Bit Numbers	Bit Mask	Description
31	0x80000000	If set to 0 then this is Standard call (pre-emptible) If set to 1 then this is a Fast Call (atomic)
30	0x40000000	If set to 0 then this is the SMC32 calling convention. If set to 1 then this is the SMC64 calling convention.
29:24	0x3F000000	

Owning Entity Number	Bit Mask	Description
0	0x00000000	ARM Architecture Calls
1	0x01000000	CPU Service Calls
2	0x02000000	SIP Service Calls
3	0x03000000	OEM Service Calls
4	0x04000000	Standard Service Calls
5-47	0x05000000 – 0x2F000000	Reserved for future use
48-49	0x30000000 – 0x31000000	Trusted Application Calls
50-63	0x32000000 – 0x3F000000	Trusted OS Calls

These ranges are further defined in section 6.

Source[2] Table 2-1 Bit usage within the SMC Function Identifier

Usage Examples by peripheral nodes

```
hdlcd@7ff50000 {
    clocks = <&scmi_clk 3>;
};

replicator@20120000 {
    power-domains = <&scmi_devpd 8>;
};
```

arm64/boot/dts/arm/juno-scmi.dtsi

Linux helpers

How do I verify everything went well?

If we don’t get failed to register error like following,
we can be sure that all the clocks/PDs registered successfully
given that it did iterate over all of them.

```
scmi-clocks scmi_dev.2: failed to register clock 292
SMCCC: SOC_ID: ARCH_SOC_ID not implemented, skipping
```

Linux commands to see and control power domains/ clocks

1. Mount the `debugfs`

```
~ # mount -t sysfs none /sys/ && mount -t debugfs none /sys/kernel/debug
~ #
```

2. change directory to `clk` or `pd`

```
~ # cd /sys/kernel/debug/clk/
/sys/kernel/debug/clk #
```

```
~ # cd /sys/kernel/debug/pm_genpd/
/sys/kernel/debug/pm_genpd #
```

3. View state of PDs

```
/sys/kernel/debug/pm_genpd # ls
ADC0                EQEP1               MMCSD0              UART6
ARM_COREPACK_0      EQEP2               MMCSD1              USB0
BOARD0              FSS0                MMCSD2              USB1
CLKDIV_0            GICSS0              MSRAM_96K0          WKUP_CLKOUT_SEL
CPSW0               GPIO0               OBSCLK0_MUX_SEL     WKUP_CPT2_AGGR0
CPT2_AGGR0          GPIO2               PBIST0              WKUP_DEEPSLEEP_
CPT2_AGGR1          GPMC0               PBIST_0             WKUP_DFTSS0
DDR16SS0            I2C0                PSC0                WKUP_DMASS0
DEBUGSS0            I2C1                ROM0                WKUP_DMASS0_DTH
DEBUGSS_WRAP0       I2C2                RTI0                WKUP_DMASS0_XLC
DMASS0              I2C3                RTI1                WKUP_GPIO0
DMASS0_BCDMA_0      LED0                STM0                WKUP_GTC0
DMASS0_PKTDMA_0     MAIN_GPIOMUX_IN     TIMER0              WKUP_GTCCLK_SEL
DPHY_TX0            MCAN0               TIMER1              WKUP_I2C0
DSS0                MCAN1               TIMER2              WKUP_OBSCLK_MUX
DSS_DSI0            MCAN2               TIMER3              WKUP_PBIST0
ECAP0               MCASP0              TIMESYNC_INTROU     WKUP_PSRAM_64K0
ECAP1               MCASP1              TRNG_DRBG_EIP76     WKUP_RTCSS0
ECAP2               MCASP2              UART0               WKUP_SMS_LITE0
ELM0                MCSPI0              UART1               WKUP_TIMER0
EPWM0               MCSPI1              UART2               WKUP_TIMER1
EPWM1               MCSPI2              UART3               WKUP_UART0
EPWM2               MCSPI3              UART4               WKUP_VTM0
EQEP0               MCU_MCU_16FF0       UART5               pm_genpd_summary
/sys/kernel/debug/pm_genpd # cd ADC/sys/kernel/debug/pm_genpd # cd ADC0/
/sys/kernel/debug/pm_genpd/ADC0 # ls
active_time      devices          pd_state         total_idle_time
current_state    idle_states      sub_domains
/sys/kernel/debug/pm_genpd/ADC0 # cat curr/sys/kernel/debug/pm_genpd/ADC0 # cat current_state
off-0
/sys/kernel/debug/pm_genpd/ADC0 #
```

4. View frequency and state of clock

```
/sys/kernel/debug/clk # ls
0_A53_0                 1_MMCSD0                4_FSS0
0_A53_1                 1_MMCSD1                4_GPMC0
0_ADC0                  1_MMCSD2                4_MCAN0
0_ARM_COREPACK_         1_OBSCLK0_MUX_S         4_MCAN1
0_BOARD0                1_PBIST_0               4_MCAN2
0_CLKDIV_0              1_PSC0                  4_MCASP0
0_CPSW0                 1_RTI0                  4_MCASP1
0_CPT2_AGGR0            1_RTI1                  4_MCASP2
0_CPT2_AGGR1            1_STM0                  4_MCSPI0
0_DDR16SS0              1_TIMER0                4_MCSPI1
0_DEBUGSS0              1_TIMER1                4_MCSPI2
0_DEBUGSS_WRAP0         1_TIMER2                4_MCSPI3
0_DMASS0_BCDMA_         1_TIMER3                4_OBSCLK0_MUX_S
0_DMASS0_PKTDMA         1_USB0                  4_RTI0
0_DPHY_TX0              1_USB1                  4_RTI1
0_DSS0                  1_WKUP_CLKOUT_S         4_TIMER0
0_DSS_DSI0              1_WKUP_DFTSS0           4_TIMER1
0_ECAP0                 1_WKUP_GPIO0            4_TIMER2
0_ECAP1                 1_WKUP_GTC0             4_TIMER3
0_ECAP2                 1_WKUP_GTCCLK_S         4_USB0
0_ELM0                  1_WKUP_I2C0             4_USB1
0_EPWM0                 1_WKUP_OBSCLK_M         4_WKUP_CLKOUT_S
0_EPWM1                 1_WKUP_RTCSS0           4_WKUP_GPIO0
0_EPWM2                 1_WKUP_TIMER0           4_WKUP_GTCCLK_S
0_EQEP0                 1_WKUP_VTM0             4_WKUP_OBSCLK_M
0_EQEP1                 20_BOARD0               4_WKUP_RTCSS0
0_EQEP2                 20_CPSW0                4_WKUP_TIMER0
0_FSS0                  20_DEBUGSS_WRAP         4_WKUP_TIMER1
0_GICSS0                20_MCASP0               50_BOARD0
```

```
/sys/kernel/debug/clk/2_GPMC0 # cat clk_rate
60000000
```

```
/sys/kernel/debug/clk # cd 2_GP/sys/kernel/debug/clk # cd 2_GPMC0/
/sys/kernel/debug/clk/2_GPMC0 # ls
clk_accuracy          clk_max_rate          clk_prepare_count
clk_duty_cycle        clk_min_rate          clk_prepare_enable
clk_enable_count      clk_notifier_count    clk_protect_count
clk_flags             clk_phase             clk_rate
/sys/kernel/debug/clk/2_GPMC0 #
```

5. Control the Clocks and PDs

Enable writeable debugfs by flag enabling flag *CLOCK_ALLOW_WRITE_DEBUGFS* for clocks and *GENPD_ALLOW_WRITE_DEBUGFS* for PDs **(*yet to be upstreamed)**

```
**********************************************************************
**     NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE             **
**                                                                  **
**  WRITEABLE clk DebugFS SUPPORT HAS BEEN ENABLED IN THIS KERNEL   **
**                                                                  **
** This means that this kernel is built to expose clk operations    **
** such as parent or rate setting, enabling, disabling, etc.        **
** to userspace, which may compromise security on your system.      **
**                                                                  **
** If you see this message and you are not debugging the            **
** kernel, report this immediately to your vendor!                  **
**                                                                  **
**     NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE             **
**********************************************************************
PM:
PM: **********************************************************************
PM: **     NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE             **
PM: **                                                                  **
PM: **  WRITEABLE power domain state DebugFS SUPPORT HAS BEEN ENABLED   **
PM: **  IN THIS KERNEL                                                  **
PM: ** This means that this kernel is built to expose pd operations     **
PM: ** such as enabling, disabling, etc.                               **
PM: ** to userspace, which may compromise security on your system.      **
PM: **                                                                  **
PM: ** If you see this message and you are not debugging the            **
PM: ** kernel, report this immediately to your vendor!                  **
PM: **                                                                  **
PM: **     NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE NOTICE             **
PM: **********************************************************************
clk: Disabling unused clocks
```

6. Controlling PDs and Clocks

After enabling writeable DEBUG, one should be able to control

```
/sys/kernel/debug/clk/2_GPMC0 # cat clk_rate
60000000
/sys/kernel/debug/clk/2_GPMC0 # echo 200000000 > clk_rate
/sys/kernel/debug/clk/2_GPMC0 # cat clk_rate
200000000
```

```
/sys/kernel/debug/pm_genpd/MMCSD0 # cat current_state
off-0
/sys/kernel/debug/pm_genpd/MMCSD0 # echo 1 > pd_state

/sys/kernel/debug/pm_genpd/MMCSD0 # cat current_state
on
/sys/kernel/debug/pm_genpd/MMCSD0 #
```

Improvements & Limitations

SCP in ATF(Hit on Latency):

1. As we know ATF runs in EL3 which means if SCP is running inside ATF, an SMC call will be triggered by SCMI call from Linux.

2. The jump from EL1 to EL3 will and then resuming back in EL1 will add to some delay in Latency.

3. The above delay is still acceptable but problem is SMC calls are not preemptible in nature from normal world. This may lead to increased latencies beyond acceptable limits.

5. To mitigate this we can run SCP in SEL1(Trusted OS), which is preemptible from Linux. But this will limit us to use only that Trusted OS in which SCP was implemented.

Lazy initialization:

1. All the clocks and PDs gets initialized at the time of init which increases the time of boot.

2. To reduce this time, need to implement something that does initialization on the fly for accessed clock/pd i.e. lazy initialization.

3. Currently, it iterates from 0 to number of clocks or PDs exported by SCMI, so if we have large number of clocks or PDs, this will lead to increased boot timings

4. This need to be fixed from Linux, which acts as SCMI client.

Clock parents selection:

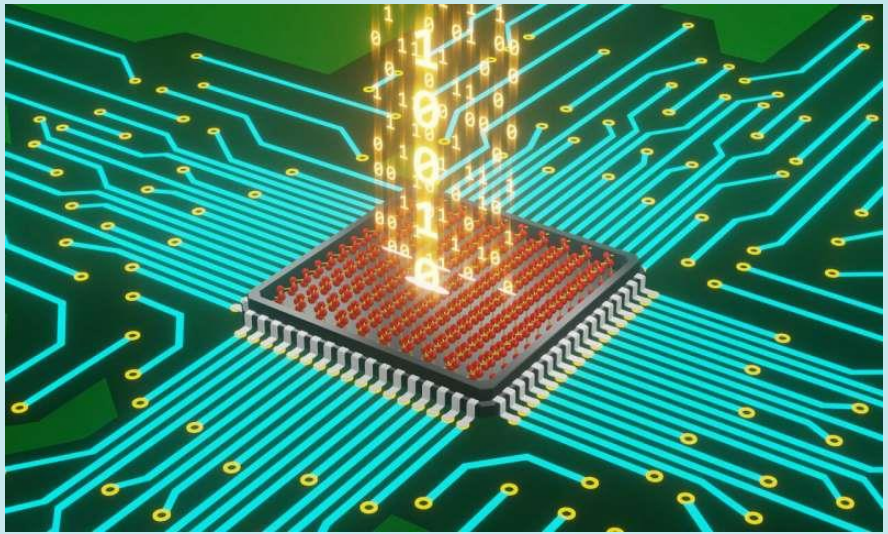
- For clocks we should be able to select the parent clock which gives us desired frequency.
- Currently, there is no support in **TF-A** that allows us to select the Parent.
- Support was added in **Linux** recently.

Extended Names:

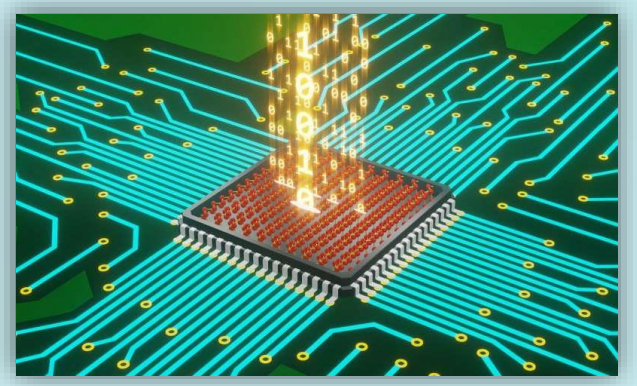
- Currently, only 16 character names are supported in **TF-A**.
- On other hand, extended names are supported in ***Linux***.
- **TF-A** needs support for extended names.

Upcoming Developments

Add SCMI v3.2 pincontrol protocol base support [3]



- Support to redirect messages from pinctrl subsystem to SCMI platform firmware.
- Set the parameter configuration and multiplexing of the pins or groups of pins.
- List the pins, groups of pins, available functions, and their association with each other.
- No support in TF-A yet.



References:

1. ARM SVC Instruction Example - SoC (s-o-c.org)
2. ARM DEN0028A: SMC Calling Convention
3. https://lore.kernel.org/all/20240415-pinctrl-scmi-v10-4-59c6e7a586ee@nxp.com/

Version info:

- DEN0056E: SCMI_v3.2
- Linux: v6.9-rc3
- TFA/ U-boot: as of 2024.04

Credits

- Texas Instruments Inc.
- The Linux Foundation.

Q&A

- Contact Information:
  - Dhruva Gole <d-gole@ti.com>
  - Kamlesh G <kamlesh@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.